Random swap EM algorithm for Gaussian mixture models

Qinpei Zhao*, Ville Hautamäki, Ismo Kärkkäinen, Pasi Fränti

Abstract

Expectation maximization (EM) algorithm is a popular way to estimate the parameters of Gaussian mixture models. Unfortunately, its performance highly depends on the initialization. We propose a random swap EM for the initialization of EM. Instead of starting from a completely new solution in each repeat as in repeated EM, we make a random perturbation on the solution before continuing EM iterations. The removal and addition in random swap are simpler and more natural than split and merge or crossover and mutation operations. The most important benefit of random swap is its simplicity and efficiency. RSEM needs only the number of swaps as a parameter in contrast to complicated parameter-setting in Genetic-based EM. We show by experiments that the proposed algorithm is 9%-63% faster in computation time compared to the repeated EM, 20%-83% faster than split and merge EM except in one case. RSEM is much faster but has lower log-likelihood than GAEM for synthetic data with a certain parameter setting. The proposed algorithm also reaches comparable result in terms of log-likelihood.

Key words: Expectation Maximization; Random Swap EM; Gaussian Mixture Model; Split and Merge EM; Genetic-based EM; data clustering;

*School of Computing, University of Eastern Finland, Finland, FI-80101, Tel: +358 132517962, Email: qinpei.zhao@uef.fi

1. Introduction

Maximum likelihood (ML) estimation of the *Gaussian mixture models* (GMMs), does not lead to a closed form solution. However, if the estimation problem is reformulated in terms of so called latent or hidden variables, a numerical gradient ascent approach can be used. As the latent variables cannot be observed directly, *expectation maximization* (EM) [1, 2] algorithm iteratively refines the ML estimate by first calculating the expectation of the posterior of the latent variables, while keeping the parameters fixed. While keeping the posteriors fixed, the algorithm then computes the maximum of the parameters. This iterative process is guaranteed to converge.

EM has two well known deficiencies. First, user needs to know in advance the number of Gaussian components. Second deficiency is that the quality depends on the initial parameters. A number of methods have been proposed to attack both problems simultaneously [3, 4]. However, such a solution needs to change the optimization cost. In general, we assume that the problem of the number of components can be solved by a validity index, and therefore, we do not consider the number of components as a parameter to be optimized.

Initial parameters are needed for the first E-step. Unfortunately, not all initial parameters lead to the same unique solution when the algorithm converges [5]. Especially for Gaussian mixture models, log-likelihood landscape is multimodal [6]. A common way to address this problem is to run EM multiple times with different randomly chosen initial parameters [5] and pick the best solution as the result. We call this variant *repeated EM* (REM). The strategy gives good stability with respect to the log-likelihood and reduces dependency on the initialization [7]. However, the solution space is searched inefficiently in REM, because after each restart it can take a long time to converge without any guarantee that it leads to an improved solution. Running time can be improved by computing in each iteration a bound on the locally optimal log-likelihood and stopping early if the bound shows no improvement [8].

Assuming that a complete restart is not necessary, search strategy based on changing only a part of the converged model can be utilized. One such strategy is to split one component into two and merge two other components [4, 9, 10, 11, 12]. A method utilizing this strategy is called *split and merge EM* (SMEM) [10], which searches systematically the best choice for the three components: one for split ($O(MN)$ operation, N is the data size and M is the number of components) and two for merge ($O(M^2N)$ operation). The choice is based on how well components match the local density of the data. Algorithm will terminate when no split and merge candidate brings improvement. Systematic approach needs to consider $O(M^3)$ triplets in total. In practice, the number of candidates searched is set lower than the number of all possible triplets.

Genetic-based EM (GAEM) [13] improves the repeated EM by considering a parallel set of solutions (populations) instead of sequential ones. Operations such as crossover, mutation and selection are applied to the population iteratively. A single-point crossover, which exchanges components between two populations is employed. Mutation selects the components with similar parameters and swaps them to random positions. A new generation of populations is finally obtained by a selection operation. There are five parameters involved in the algorithm. In general, GAEM can achieve a good result by a proper set of parameters.

Some other algorithmic strategies employed to escape a local maximum are: competitive learning [4], incremental clustering implemented in *greedy EM* (GEM) [14], stochastic variants such as *stochastic EM* (SEM) [15] and *Monte Carlo EM* (MCEM) [16].

In this work, we use randomization instead of systematic search to select the component. Preliminary results of the proposed method were published in [17, 18]. In the proposed algorithm, *random swap EM* (RSEM), replaces the split and merge -operations by more general addition and removal -operations. Proposed operations are simple and efficient. Removing a component, which is an $O(1)$ time operation, is more straightforward than merging and only one component is involved. Creation of a new component is also simpler than splitting a component, where split is usually ill-posed (i.e., more variables than equations). GAEM has five parameters, all of which affect the running time and performance. Proposed method is thus simpler and easier to adapt to different datasets and applications.

In RSEM, randomly selected component is swapped to a new location in the feature space and the weight and covariance matrices are updated. The time complexity is $O(NM)$, which is the same as one EM iteration. Even though more iterations are needed by random swap approach due to its trial-and-error nature, the total number of candidates is significantly less than by systematic search such as SMEM or repeated EM. After the swap is performed, EM is iterated until convergence. New solution is accepted only if it improves the previous one. In principle, RSEM algorithm terminates when none of the possible NM swaps result in an improved solution [19]. However, a fixed number of swaps is sufficient in practice.

2. EM algorithm and its Variants

In this section, we first describe the existing methods that are compared to the proposed method, which is presented in Section 3.

2.1. EM algorithm

EM algorithm can be used to estimate *maximum likelihood* (ML) parameters of many different types of parametric densities. For GMMs, the goal is to maximize the following log-likelihood:

$$L(\Theta) = \log p(\boldsymbol{X}|\Theta) = \sum_{i=1}^{N} \log \sum_{j=1}^{M} \alpha_j \mathcal{N}(\boldsymbol{x}_i|\Theta_j), \tag{1}$$

where $\mathcal{N}(.|.)$ is Gaussian distribution, $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ is the observed d-dimensional data-set of N vectors, Θ is the GMM and $\Theta_j = (\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ are the mean vector and covariance matrix of the jth Gaussian, respectively. Finally, α_j is the mixture weight of the jth component. The parameters α_j must satisfy the following constraints:

$$\sum_{j=1}^{M} \alpha_j = 1, \quad \text{and,} \quad \alpha_j \geq 0, \quad j = 1, ..., M. \tag{2}$$

Unfortunately, closed-form solution of the (1) is not possible [1], since it contains the log of the sum. Maximization is then performed on the expectation of the complete-data log-likelihood, given posterior density of the latent variables [1]. This function is usually called the Q-function, and can be written in a concrete form for Gaussian mixtures as:

$$Q(\Theta, \Theta^{\mathrm{t}-1}) = \sum_{i=1}^{N} \sum_{j=1}^{M} \tau_{ij} \left\{ \log \alpha_j + \log \mathcal{N}(\boldsymbol{x}_i|\Theta_j) \right\}. \tag{3}$$

$\Theta^{\mathrm{t}-1}$ are parameters estimated in the previous iteration. Maximization of Eq. (3), in terms of Θ can be performed easily, by keeping the posterior probabilities τ_{ij} fixed. Then, given estimated parameters, posterior probability of $\boldsymbol{x}_i$ from component j, τ_{ij} can be calculated as follows:

$$\tau_{ij} = \frac{\mathcal{N}(\boldsymbol{x}_i|\Theta_j)\alpha_j}{\sum_{l=1}^{M} \mathcal{N}(\boldsymbol{x}_i|\Theta_l)\alpha_l} \tag{4}$$

To find an initial set of parameters in EM algorithm, one possibility is to randomly select mean vectors and set equal weights and whole data covariance matrix for all components [20]. A more common practice is to first run k-means on the dataset to get hard partitioning. The initial mean vectors are directly the cluster centroids, partition covariance is the component covariance matrix and proportion of vectors in each partition is the component weight. Several short runs of k-means starting with random initial solutions each followed by a long run of EM is recommended in [7].

EM suffers from the local maximum problem [6]. A standard solution for the initialization problem (REM) is to repeat random initializations with k-means followed by EM [7]. The best performing solution, in terms of log-likelihood, is retained. This introduces a new parameter, the number of repeats. From the linearity of expectation, it is expected that the number of EM iterations in REM is multiplied by the number of repetitions. It means that the model quality can be improved by increasing the number of repetitions, but at the cost of linearly increasing the processing time.

2.2. Split-and-Merge EM

One strategy to overcome the sensitivity to initialization of EM algorithm is to identify parts of the solution that do not fit well to the data, and revise the solution by making local changes. When working in the component domain, we can change the solution by splitting a component into two and by merging two components into one. *Split and merge EM* (SMEM) [10] makes a systematic search through all possibilities for split and merge after which the algorithm selects the best candidates and performs the operations.

SMEM algorithm searches among the candidates composed of combinations of all components i, j and k until the likelihood value improves. The candidates are sorted by the merge and split criteria. Merge criterion is based on the correlation of posterior probabilities of components i and j. The split criterion is based on the Kullback-Leibler divergence between component k and the local data density.

$$\begin{aligned}\mathrm{JMerge}(i,j) &= \frac{\tau_i(\Theta)^T\tau_j(\Theta)}{||\tau_i(\Theta)||||\tau_j(\Theta)||}\\ \mathrm{JSplit}(k) &= \int f_k(\boldsymbol{X},\theta_k)\log\frac{f_k(\boldsymbol{X},\theta_k)}{p_k(\boldsymbol{X},\theta_k)}dx\end{aligned} \tag{5}$$

where, $\tau_i(\Theta) = (\tau_{1i}(\Theta), ..., \tau_{Ni}(\Theta))$ is an N-dimensional vector consisting of the posterior probabilities for the ith component. T denotes the transpose operation and $1 < k \neq i \neq j < M$. The $f_k(\boldsymbol{X},\theta_k)$ is the local data density around the component k and the $p_k(\boldsymbol{X},\theta_k)$ is the empirical distribution. The merged components are combined linearly and the split component is split by adding constant movements on the original parameters. Then a partial EM step is performed on the merge and split candidate.

The original acceptance rule, line 7 in Algorithm 1, used the Q-function, instead of $L(\Theta)$ [10]. However, it was found in [21] that by doing so the global maximum might be accidentally rejected. In our experiments, we therefore use improvement of the log-likelihood as the acceptance rule.

A practical problem of split and merge approach is that the split and merge operations are not straightforward to design. The assumption behind split-and-merge approach is that only the components of the triplet (i, j, k) are affected and the rest of the model is unchanged. Merge operation has a closed-form solution when we assume that the distributions are Gaussian. However, it is not possible to find a unique solution to the problem of splitting one component into two.

One alternative was proposed in [12], where one randomly selected singular value decomposition basis vector of the covariance matrix is used to compute two new covariance matrices. It is also used in combination with the original mean vector to generate two new mean vectors.

```
Input: Data Set $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$
Output: Parameters $\Theta = \{\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ and log-likelihood $L(\Theta)$
1  [$\Theta_0$, $L(\Theta_0)$] $\leftarrow$ EM($X$);
2  while candidates left to process do
3      Sort candidates $(i, j, k)_{C_{\max}}$ by JMerge and JSplit (equation 5);
4      for $c = 1$ to $C_{\max}$ do
5          [$\Theta^{'}$, $L(\Theta^{'})$] $\leftarrow$ partialEM($(i, j, k)_c$);
6          [$\Theta^{*}$, $L(\Theta^{*})$] $\leftarrow$ EM($\boldsymbol{X}$, $\Theta^{'}$);
7          if ($L(\Theta^{*}) > L(\Theta)$) then
8              $\Theta = \Theta^{*}$; $L(\Theta) = L(\Theta^{*})$;
9          end
10     end
11 end
12 return $\Theta, L(\Theta)$
```

Algorithm 1: SMEM algorithm

Furthermore, due to the split and merge operations, $C_{\max} = M(M-1)(M-2)/2$ candidate triplets are generated. A systematic search through all possible triplets leads to $O(M^4 N I_{\text{EM}})$ time complexity, where I_{EM} is the number of EM iterations needed to reach convergence after perturbation. Final processing time can be reduced by considering only top $C_{\max}$ candidates. In [10], $C_{\max}$ was set to five. We first experimentally find suitable $C_{\max}$ before comparing SMEM to other methods.

```
   Input: Data Set $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, $I_{EM}$, $I_g$, $I_p$, $p_c$, $t_{corr}$
   Output: Parameters $\Theta = \{\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ and log-likelihood $L(\Theta)$
1  [$\Theta_p[I_p]$, $L[I_p]$] $\leftarrow$ Initialization($\boldsymbol{X}$);
2  for GAEM-iteration=$1$ to $I_g$ do
3      [$\Theta_p[I_p]$, $L_1[I_p]$] $\leftarrow$ EM($\Theta_p[I_p]$, $I_{EM}$);
4      $\Theta_c[H]$ $\leftarrow$ crossover($\Theta_p[I_p]$, $p_c$); $H = I_p * p_c$;
5      [$\Theta_c[H]$, $L_2[H]$] $\leftarrow$ EM($\Theta_c[H]$, $I_{EM}$);
6      [$\Theta_s[I_p]$, $L_1(\Theta_s[I_p])$] $\leftarrow$ Select ($\Theta_p[I_p]$, $\Theta_c[H]$, $L_1[I_p]$, $L_2[H]$);
7      $\Theta_s[I_p]$ $\leftarrow$ mutation($\Theta_s[I_p]$, $t_{corr}$);
8      $\Theta_p[I_p]$ $\leftarrow$ $\Theta_s[I_p]$;
9  end
10 execute lines 3 to 6 once;
11 [$\Theta$, $L$] $\leftarrow$ EM($\Theta_s[best]$, $I_{EM}$);
12 return $\Theta$, $L$
```

Algorithm 2: GAEM algorithm

2.3. Genetic-based EM

Genetic-based EM (GAEM) for learning Gaussian mixture models is proposed in [13]. Original design of GAEM includes the model selection. However, number of components M is left as a user defined parameter in our task definition. So, we have modified the algorithm by keeping M fixed and removing the part where decision regarding M is made. Also, instead of MDL criterion we use log-likelihood during the selection in Algorithm 2.

In GAEM, the single point crossover operator selects a component index. First child gets components before the index from first parent and from the index onwards from the second parent, and vice versa for the second child. Mutation operator selects components that model the data points similarly by using posterior probabilities (i.e., $JMerge(i, j)$). If there is a correlation above a given parameter limit, the components are moved to random positions. New generation is selected from parent and child populations.

There are two deficiencies in GAEM. One is that the algorithm involves multiple solutions (population). When the population size (I_p) is large enough, a good result is achieved but it increases the running time linearly. The other one is the parameters. For crossover, mutation and selection steps, parameters are involved. In crossover, a probability p_c determines the number of offsprings after crossover. A threshold for correlation coefficient t_{corr} between components is set for mutation. There are also parameters for GAEM iterations I_g and EM iterations I_{EM}.

3. Random Swap EM

The idea of the *random swap EM* (RSEM) algorithm is to alternate between simple perturbation to the solution by random swap and convergence towards nearest optimum by the EM algorithm. A random swap consists of removal and addition operations.

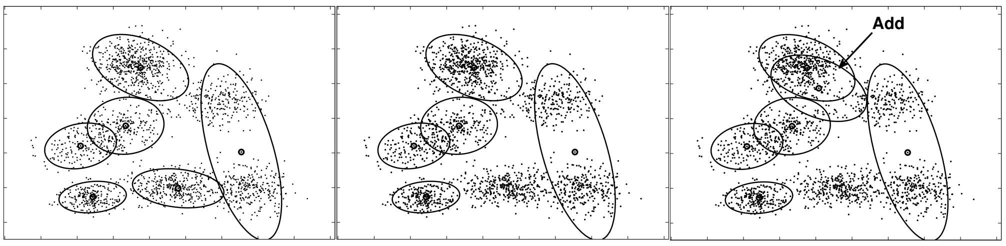
RSEM is presented in Algorithm 3. The initialization is performed as in the EM algorithm, described in Section 2.1. After the solution has been initialized, we perform t random swap iterations (called RS-iterations). During each iteration, a component is removed, a new one is added and the resulting solution is converged towards nearest optimum using EM algorithm. The best solution, in terms of log-likelihood, is maintained as the starting point for the subsequent RS-iteration.

```
   Input: Data Set $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$
   Output: Parameters $\Theta = \{\boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ and log-likelihood $L(\Theta)$
 1 $[\Theta_0, L(\Theta_0)] \leftarrow$ Initialization($\boldsymbol{X}$);
 2 for RS-iteration=$1$ to $t$ do
 3     $r = \mathrm{U}(1, M)$, remove $r$th component;
 4     $p = \mathrm{U}(1, N)$, add at $p$th position (see equation 7);
 5     normalize weights $\boldsymbol{\alpha}$ to sum to 1;
 6     new parameters $\Theta^{\mathrm{s}} = \{\alpha^{\mathrm{s}}, \boldsymbol{\mu}^{\mathrm{s}}, \boldsymbol{\Sigma}^{\mathrm{s}}\}$;
 7     $[\Theta^{\mathrm{st}}, L(\Theta^{\mathrm{st}})] \leftarrow$ EM($\boldsymbol{X}, \Theta^{\mathrm{s}}$);
 8     if $L(\Theta^{\mathrm{st}}) > L(\Theta)$ then
 9         $\Theta = \Theta^{\mathrm{st}}$;
10         $L(\Theta) = L(\Theta^{\mathrm{st}})$;
11     end
12 end
13 return $\Theta, L(\Theta)$
```

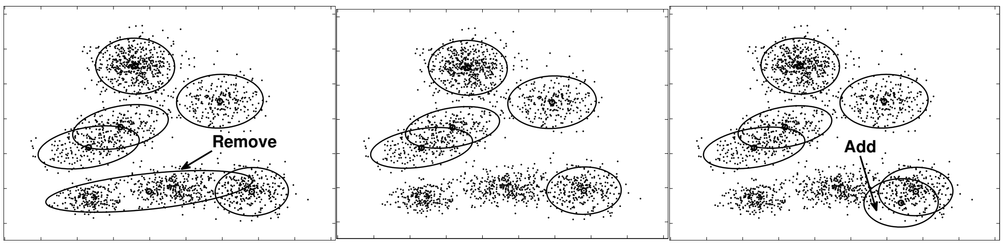
Algorithm 3: RSEM algorithm

The removal operation is done by selecting a component r randomly among M components from uniform distribution, $r = \mathrm{U}(1, M)$. This is a constant-time operation.

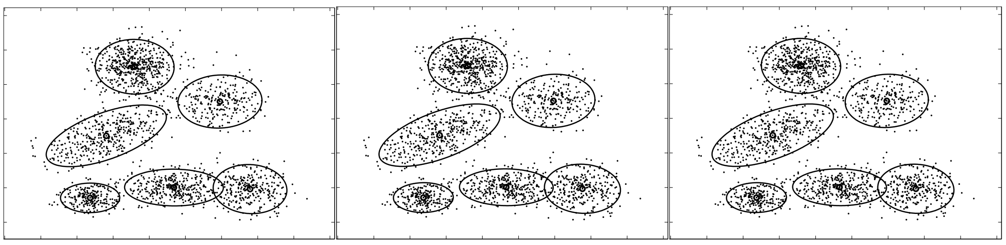
The location of the new component is decided by selecting one data point, $\boldsymbol{x}_p, p = \mathrm{U}(1, N)$ and setting it as the mean vector of the new component. The new component is therefore more likely to be placed in areas of high point density, such as cluster centers, than areas of low point density.



(a) Initial solution by k-means (b) Remove a component (c) Add a component



(d) Convergence by EM (e) Remove a component (f) Add a component



(g) Convergence by EM (h) After 10 iterations of RSEM (i) Ground truth Gaussians

Figure 1: Result by RSEM for a two-dimensional Gaussian mixture density estimation problem. (a) An initial solution by 10 runs of k-means, (b)-(c) Removal and addition operation for the 1^{st} iteration , (d) Convergence by EM, (e)-(g) The procedures on the 3^{rd} iteration, (h) The final result by RSEM with 10 iterations, (i) Ground-truth Gaussians.

The best solution found so far, in terms of log-likelihood, is always used as the starting point for the next iteration. If a swap and EM iterations fail to produce a better solution than the starting point, the new solution is discarded. Swap will decrease the log-likelihood of the solution but it can also change the solution so that iterating EM will move it towards different optimum.

The technique has been successfully applied to clustering with centroid model [22, 23, 24]. We observed that the effect of a bad initialization is diminished when random swap is used. We therefore expect random swap to yield good results with Gaussian mixture models, too.

The solution is fine-tuned with EM algorithm, so reasonable values for the weight and covariance matrix are sufficient. Suppose the current likelihood function $L(\Theta^{\mathrm{t}})$ at RS-iteration t is obtained by EM. Let r be the component selected for removal, and keep the rest of the components unchanged. The posterior probability is updated as follows:

$$\tau_{ij}^{\mathrm{s}} = \frac{\alpha_j^t \mathcal{N}(\boldsymbol{x}_i|\Theta_j^t)}{\sum_{l=1,l\neq r}^{M} \alpha_l^t \mathcal{N}(\boldsymbol{x}_i|\Theta_l^t)} \tag{6}$$

The equations for the new parameters of the r^{th} component are:

$$\begin{aligned}
&\boldsymbol{\mu}_r^{\mathrm{s}} = \boldsymbol{x}_p \\
&\alpha_r^{\mathrm{s}} = \alpha_r^{\mathrm{t}} \quad or \quad \alpha_r^{\mathrm{s}} = \sum_{l=1,l\neq r}^{M} \left(\sum_{i=1}^{N} \tau_{il}^{\mathrm{s}} \right) \alpha_l^{\mathrm{t}} \\
&\boldsymbol{\Sigma}_r^{\mathrm{s}} = \boldsymbol{\Sigma}_r^{\mathrm{t}} \quad or \quad \boldsymbol{\Sigma}_r^{\mathrm{s}} = \sum_{k=1,k\neq r}^{M} \left(\sum_{i=1}^{N} \tau_{ik}^{\mathrm{s}} \right) \boldsymbol{\Sigma}_k
\end{aligned} \tag{7}$$

In order to retain a valid Gaussian mixture model after the swap operation, weights $\alpha_i, 1 \leq i \leq M$ are normalized so that they sum up to 1. The time complexity of the addition operation is linear with respect to the model size M. After each swap, the new parameters Θ^{s} are set as initial solutions for EM. After EM has converged, we get a new likelihood value $L(\Theta^{\mathrm{st}})$ and we compute $\Delta L = L(\Theta^{\mathrm{st}}) - L(\Theta^{\mathrm{t}})$, If the difference is positive, the new parameter estimate replaces the previous best solution. Otherwise the new parameter estimate is discarded. This process is repeated until all possible swap pairs are tried out and none is left to improve the solution. However, as a practical matter we restrict the total number of swaps to a user selectable number of RS iterations t. An example of RSEM algorithm operating on data is illustrated in Fig. 1.

To ensure a good solution, the number of iterations t for random swap should be set large enough so that there are enough successful swaps. Given the number of components M, the probability of selecting a component to be removed is $1/M$. The probability of selecting a point to be added is also $1/M$. Only if the point is inside one cluster, it will be a successful addition because EM can fine-tune the location even after then. Therefore it is not necessary to find near-optimal location during creation of a component. For a good swap to occur, a badly-placed component must be chosen and a location from the area where the component needs to move must also be chosen. Hence the probability of a single good swap is at least $1/M^2$, and $t > M^2$.

4. Summary of Iterative Methods

4.1. Comparing REM and RSEM

RSEM is faster than REM if it converges with fewer iterations after a swap. We prove in [18] that the increment of Q-function value by randomly swapping a component in RSEM is greater than that by a random restart on all components in REM, which leads to the fact that processing time of RSEM is less than REM for reaching the optimal result. We will approximate log-likelihood by the Q-function as in [8].

Theorem 4.1. *A random swap limits $Q(\Theta^s, \Theta^{t-1}) - Q(\Theta^t, \Theta^{t-1})$ into the lower* *and upper bounds of* $[-\frac{N\alpha_r^t}{2}d, \frac{N\alpha_r^t}{2}d]$*, where* d *is the Mahalanobis distance between the swapped centroids* μ_r^t *and* μ_r^s*.*

Theorem 4.2. *For REM and RSEM, if* $d < \frac{1}{3}$*, the probability of* $Q(\Theta^s, \Theta^{t-1}) - Q(\Theta^t, \Theta^{t-1}) > Q(\Theta, \Theta^{t-1}) - Q(\Theta^t, \Theta^{t-1})$ *is* 1*. If* $d \geq \frac{1}{3}$*, the probability is* $\frac{1}{2} + \frac{1}{6d}$*.*

We see that the farther the new component is from the original, the closer to $P = 1/2$ we approach. However, REM will not have a higher probability than RSEM to reach a high Q-function value.

4.2. Comparison of time complexities

The time complexities of the algorithms are shown in Table. 1. M and N are the number of clusters and data vectors, respectively. S represents the number of REM repetitions, the number of RSEM swaps and the number of SMEM iterations with improvement. Parameters I_1, I_2 and I_3 are the iteration counts of EM convergence in the algorithms and C in SMEM indicates the number of candidates, which is set $C = 20$ in our experiments. Parameter I_g is the number of generations, I_{EM} is the number of EM iteration used in GAEM and I_p is the population size.

REM and RSEM have similar strategies. The difference is in the number of EM iterations to converge in both methods. Since not every run of EM contributes to the final result in REM, the proposed RSEM algorithm, which changes only a part of the solution, achieves better or same result faster than REM. This is shown theoretically in [18] and experimentally in Section 5. For SMEM, the number of SMEM iterations with improvement S takes a major role in the time complexity of SMEM. It highly depends on the size of search space caused by the the number of candidates C. RSEM is faster than SMEM when $I_2 \leq M + CI_3$. The merge

Table 1: Time complexity analysis on the methods.

			total
REM	EM	$O(I_1MN)$	$O(SI_1MN)$
SMEM	merge	$O(MN + M^2N)$	$O(S(M^2N + CI_3MN))$
	split	$O(MN + N\log N)$	
	EM	$O(3N)$	
GAEM	mutation	$O(M^2)$	$O(I_gI_p^2I_{EM}MN)$
	crossover	$O(I_pM)$	
	EM	$O(I_p^2MNI_{EM})$	
RSEM	removal	$O(1)$	$O(SI_2MN)$
	addition	$O(MN)$	
	EM	$O(I_2MN)$	

operation in SMEM takes much more time than removal in RSEM. Thus, RSEM is faster than SMEM in most cases. In GAEM, number of generation I_g plays a similar role as S, then RSEM is faster than GAEM if an average EM iterations are less than $I_p^2I_{EM}$. On the other hand, we can also restrict EM iterations in RSEM to I_{EM}, then extra computations caused by GAEM is quadratic to population size.

5. Experimental Results

We tested the algorithms[1] using both synthetic and real data sets from various sources summarized in Table 2. We divide the sets into two categories. The first category is synthetic data sets. These are fairly small and contain a known number of clusters. In the tests, we match the number of components with the number

[1]`http://cs.joensuu.fi/sipu/soft/`

of clusters whenever the number of clusters is known. The second category is large data sets obtained from UCI Machine Learning Repository [25]. We set the number of components to 15 for CM and 20 for CT.

Table 2: Attributes of the data sets used in our experiments.

Data sets	Name	Dimension	Data Size	No. of Clusters
Synthetic	S1-S4 [26]	2	5000	15
	R15 [27]	2	600	15
Real	CM [28]	9	68040	15
	CT [28]	16	68040	20

In all experiments Gaussian mixture models are restricted to diagonal covariance matrices. The baseline algorithm is the REM algorithm. Initialization of the GMM for each repetition is described in Section 2.1. RSEM is given one random initial solution and the same number of RS-iterations is performed as the number of random solutions given to REM. The EM algorithm or partial EM algorithm is allowed to iterate until convergence (threshold = $1.53e-05$), except in GAEM, $I_{EM} = 3$.

The number of candidates $C_{\max}$ considered in each SMEM round is fixed to 20 as it seems to provide the best accuracy and processing time trade-off (see supplementary[2]). Increasing the number of candidates closer to maximum $O(M^3)$ does not improve the accuracy at all. SMEM algorithm immediately accepts a candidate that results in a better solution. When none of the $C_{\max}$ candidates result in improvement, the algorithm stops.

[2]http://cs.joensuu.fi/~zhao/Software/supplementary1.pdf

For GAEM, both I_p and I_g affect the running time. However, the result in terms of log-likelihood depends more on I_p. An experiment on different combinations of I_p and I_g on data S2 is conducted (see the supplementary file). The number of generations helps little to improve the log-likelihood, which however brings high running time. Thus, we select $I_g = 10$. The population size I_p affects the result clearly. It seems the log-likelihood is stable when $I_p > 20$ for S2. However, since the running time of GAEM (proportional to I_p^2) depends highly on I_p, we choose $I_p = 15$ to reduce the running time. The crossover probability $p_c = 0.8$ and $t_{corr} = 0.95$ following the setting in [13].

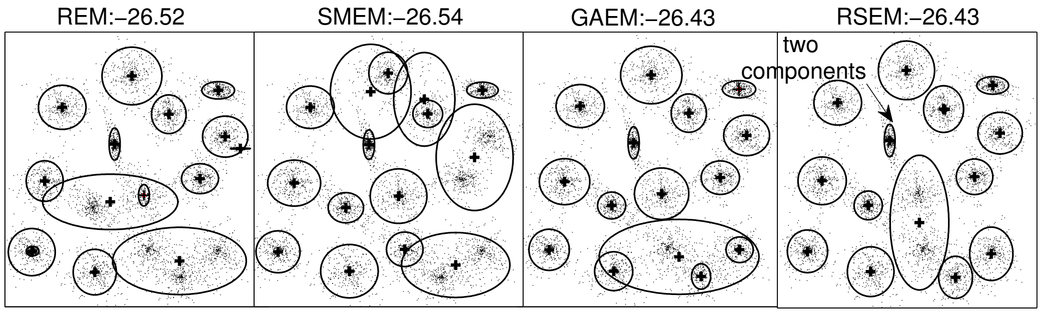


Figure 2: Gaussian models on data S2 estimated from EM variants.

We demonstrate the Gaussian models estimated from REM, SMEM, GAEM and RSEM on data set S2 in Fig. 2. The experiment is conducted by 20 repetitions. The average among them in terms of log-likelihood is shown. The models are displayed as ellipses. REM and SMEM are clearly worse in parameter estimation than GAEM and RSEM.

For S1 to S4, ground-truth distributions are available. For comparing the GMMs obtained from different EM variants, we calculate the squared Euclidean distance between estimated and ground-truth GMMs using the closed-form solution in [29]. The distance values are the average of 50 results. There are two out of four cases that RSEM is closer to ground-truth than competing methods even though log-likelihood is the best in all cases. It implies that in terms of parameter estimation by likelihood is not always a good proxy. The goal metric, however in the present work is log-likelihood.

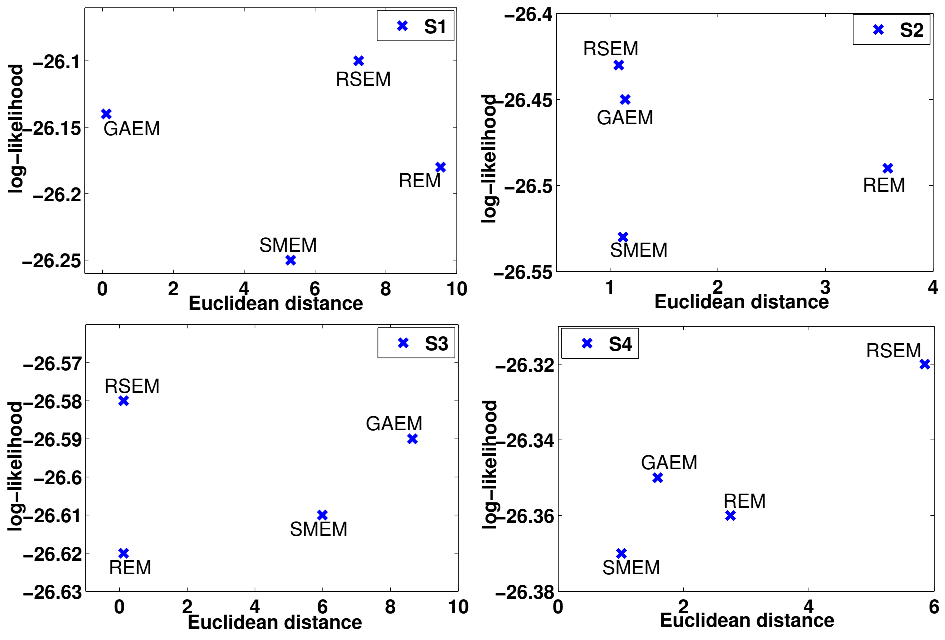


Figure 3: Squared Euclidean distance between ground-truth GMM and estimated solutions vs. log-likelihood values.

To obtain robust estimates of average log-likelihood and CPU time values, each algorithm is repeated 50 times. A summary on the mean log-likelihood values is presented in Table 3 and processing time in Table 4. Statistical tests run on the distributions of log-likelihood values and processing times showed that the processing time follows Gaussian distribution while log-likelihoods do not. Furthermore, the shapes of the log-likelihood distributions differ from each other.

Hence statistical significance tests such as t-test or normal rank-sum test can not be used for log-likelihoods. Thus, we performed t-test only on the processing time of RSEM and other three methods (REM, SMEM and GAEM) respectively to emphasize that RSEM is significantly faster than the EM variants with comparable or better log-likelihood. We use an asterisk (*,$p < 0.05$) to indicate the significant difference between RSEM and other EM variants.

Table 3: Summary of the mean log-likelihood values.

	S1	S2	S3	S4	R15	CM15	CT20
REM	-26.20	-26.51	-26.63	-26.37	-6.48	-10.34	-3.64
SMEM	-26.25	-26.53	-26.61	-26.38	-6.57	-10.35	-3.65
GAEM	**-26.11**	**-26. 43**	**-26.59**	-26.34	**-6.35**	-10.35	-3.65
RSEM	-26.15	-26.45	-26.60	**-26.34**	-6.43	**-10.33**	**-3.63**

Table 4: Summary of the mean processing times (seconds).

	S1	S2	S3	S4	R15	CM15	CT20
REM	3.18*	3.94*	4.59*	4.07*	0.32*	794*	2551*
SMEM	2.29*	2.80*	3.34*	4.38*	0.29*	2267*	961
GAEM	7.09*	6.82*	6.45*	6.59*	1.13*	**157**	**315**
RSEM	**1.27**	**1.66**	**1.71**	**1.70**	**0.21**	355	1568

In processing time SMEM can vary greatly. The variance mainly comes from the $C_{\max}$ candidates. The algorithm stops if there is no improvement among the candidates, which decreases the running time in some cases. This is also reflected in log-likelihoods for CT data set. SMEM is capable of improving the initial solutions according to log-likelihood, but the effort needed varies greatly, resulting in large variation in running times. The other algorithms are not affected much by the data set. Difference in running time between REM and RSEM is explained by the need to improve the entire model in REM versus the smaller changes in RSEM.

GAEM has good performance in terms of log-likelihood, however, it is much slower than RSEM for synthetic data. For real data, the running time is faster than RSEM, however the log-likelihood is worse. This is a major difficulty in using GAEM in practical applications. How to set parameters for a new dataset in such way that quality of the solution is maintained while processing time is kept in control. In contrast, RSEM offers simplicity to users. If processing time is not an issue, RSEM can be run until convergence, and then no parameter is required.

6. Conclusions

We proposed a random swap EM algorithm in order to get rid of the tendency of the standard EM algorithm to get stuck in a local maximum. The proposed RSEM indicates that it is not necessary to start from the beginning in each restart as it does in the repeated EM. The RSEM is also shown to be simpler and more efficient than other EM variants. The removal and addition operations in RSEM are more general and simpler than split and merge operations in SMEM. They use less parameters than crossover and mutation in GAEM, where crossover involves two populations at a time and a criterion is needed in mutation. Comparing the proposed algorithm to the REM, we found that RSEM reached higher or comparable level of log-likelihood 9%-63% faster, which was proved by a bound derived from formulas. RSEM is also easier to implement and more efficient than the split-and-merge EM (20%-83% faster). Genetic EM has good performance, however, the complicated parameter setting makes it less useful in practice.

The number of swaps is a key parameter in the proposed method, which decides the performance of RSEM. As a future work, we plan to investigate ways to automatically select the number of swaps, as well as theoretical support for random swap strategy in Gaussian mixture models.

References

[1] Bishop, C.: Pattern Recognition and Machine Learning. Springer Verlag (2006)

[2] Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. Journal of Royal Statistical Society B **39** (1977) 1–38

[3] Figueiredo, M., Jain, A.: Unsupervised learning of finite mixture models. IEEE Transactions on Pattern Analysis and Machine Intelligence (2002) 381–396

[4] Zhang, B., Zhang, C., Yi, X.: Competitive EM algorithm for finite mixture models. Pattern Recognition **37** (2004) 131–144

[5] McLachlan, G., Peel, D.: Finite Mixture Models. John Wiley &Sons (2000)

[6] Mclachlan, G., Krishnan, T.: The EM algorithm and extensions. John Wiley &Sons (1996)

[7] Biernacki, G., Celeux, C., Govaert, G.: Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models. Computational Statistics and Data Analysis **41** (2003) 561–575

[8] Zhang, Z., Dai, B., Tung, A.: Estimating local optimums in EM algorithm over Gaussian Mixture Model. Proceedings of the 25th International Conference on Machine Learning (2008) 1240–1247

[9] Li, Y., Li, L.: A split and merge EM algorithm for color image segmentation. IEEE Intl. Conf. on Intelligent Computing and Intelligent Systems (2009) 395–399

[10] Udea, N., Nakano, R., Gharhamani, Z., Hinton, G.: SMEM algorithm for mixture models. Neural Computation **12** (2000) 2109–2128

[11] Wang, H., Luo, B., Zhang, Z.: Estimation for the number of components in a mixture model using stepwise split-and-merge EM algorithm. Pattern Recognition Letters **25**(16) (2004) 1799–1809

[12] Zhang, Z., Chen, C., Sun, J., Chan, K.: EM algorithms for Gaussian mixtures with split-and-merge operation. Pattern Recognition **36**(9) (2003) 1973–1983

[13] Pernkopf, F., Bouchaffra, D.: Genetic-based em algorithm for learning gaussian mixture models. IEEE Transactions on Pattern Analysis and Machine Intelligence **27**(8) (2005) 1344–1348

[14] Verbeek, J., Vlassis, N., Krose, B.: Efficient greedy learning of Gaussian mixture models. Neural Computation **15**(12) (2003) 469–485

[15] Celeux, G., Diebolt, J.: The SEM algorithm: a probabilistic teacher algorithm derived from the EM algorithm for the mixture problem. Computational Statistics Quaterly **2** (1985) 73–82

[16] Wei, G., Tanner, M.: A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. Journal of the American Statistical Association **85** (1990) 699–704

[17] Zhao, Q., Hautamäki, V., Kärkkäinen, I., Fränti, P.: Randon swap EM algorithm for finite mixture models in image segmentation. In: Proc. IEEE Int. Conf. on Image Processing, Cairo, Egypt (November 2009) 2397–2400

[18] Zhao, Q., Hautamäki, V., Fränti, P.: Rsem: An accelerated algorithm on repeated em. 2011 Sixth International Conference on Image and Graphics (2011) 135–140

[19] Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C., Silverman, R., Wu, A.Y.: A local search approximation algorithm for k-means clustering. Computational Geometry: Theory and Aplications **28** (2004) 89–112

[20] Figueiredo, M., Jain, A.: Unsupervised learning of Finite Mixture Models. IEEE Transactions on Pattern Analysis and Machine Intelligence **24**(3) (2002) 381–396

[21] Minagawa, A., Tagawa, N., Tanaka, T.: SMEM algorithm is not fully compatible with maximum-likelihood framework. Neural Computation **14**(6) (2002) 1261–1266

[22] Fränti, P., Kivijärvi, J.: Randomized local search algorithm for the clustering problem. Pattern Analysis and Applications **3**(4) (2000) 358–369

[23] Fränti, P., Virmajoki, O., Hautamäki, V.: Probabilistic clustering by random swap algorithm. 19th Int. Conf. on Pattern Recognition **55**(4) (2008) 287–314

[24] Merz, P.: An iterated local search approach for minimum sum-of-squares clustering. Advances in Intelligent Data Analysis V (2003) 1–38

[25] Asuncion, A., Newman, D.: UCI machine learning repository. `http://archive.ics.uci.edu/ml/` (2007)

[26] Fränti, P.: Clustering data sets. `http://cs.joensuu.fi/sipu/datasets/` (2009)

[27] Veenman, C., Reinders, M., Backer, E.: A maximum variance cluster algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence **24**(9) (2002) 1273–1280

[28] Ortega, M., Rui, Y., Chakrabati, K., Porkaew, K., Mehrotra, S., Huang, T.: Supporting ranked boolean similarity queries in MARS. IEEE Transactions on Knowledge and Data Engineering **10**(6) (1998) 905–925

[29] Helen, M., Virtanen, T.: Query by example of audio signals using euclidean distance between gaussian mixture models. ICASSP'07 (2007) 225–228